BLSURF - Mesh Generator for Composite Parametric Surfaces - User's Manual

Patrick Laug, Houman Borouchaki

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

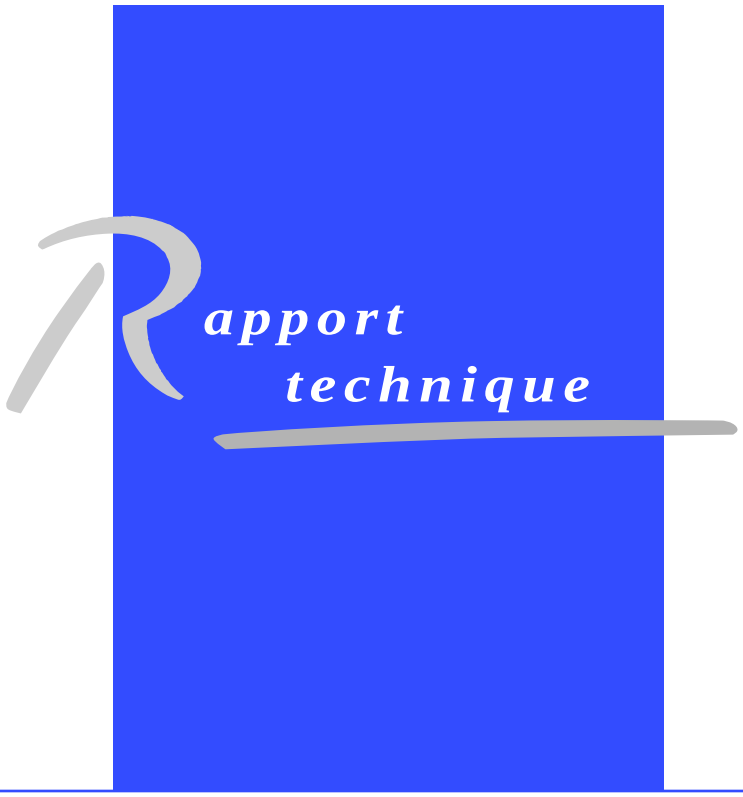
BLSURF – Mesh Generator for Composite Parametric Surfaces – User's Manual

Patrick Laug , Houman Borouchaki

No 0235

November 2, 1999

THÈME 4



ISSN 0249-0803

BLSURF – Mesh Generator for Composite Parametric Surfaces – User's Manual

Patrick Laug* , Houman Borouchaki**

Thème 4 — Simulation et optimisation
de systèmes complexes
Projet Gamma

Rapport technique n ° 0235 — November 2, 1999 — 48 pages

Abstract: This technical report describes BLSURF, an automatic surface mesh generator. This software creates the mesh of a composite parametric surface while conforming to a prescribed size map. To this end, analytical functions defining the surface, as well as their first and possibly second derivatives, must be externally provided (for instance by a CAD system). Also, each patch must be described by giving its 2D parametric domain. The method used is based on an incremental Delaunay approach adapted to a Riemannian metric.

Key-words: surface mesh, composite parametric surface, Riemannian metric, Delaunay triangulation, advancing front approach.

(Résumé : tsvp)

* INRIA, Institut National de Recherche en Informatique et en Automatique, UR de Rocquencourt, BP 105, 78153 Le Chesnay cedex, France. Email: Patrick.Laug@inria.fr.

** UTT, Université de Technologie de Troyes, 12 rue Marie Curie, BP 2060, 10010 Troyes cedex, France. Email: Houman.Borouchaki@univ-troyes.fr.

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
Téléphone : 01 39 63 55 11 - International : +33 1 39 63 55 11
Télécopie : (33) 01 39 63 53 30 - International : +33 1 39 63 53 30

BLSURF – Mailleur de surfaces composées de carreaux paramétrés – Manuel d'utilisation

Résumé : Ce rapport technique décrit les fonctionnalités du mailleur automatique surfacique BLSURF. Ce logiciel génère le maillage d'une surface composée de plusieurs carreaux paramétrés, en respectant un champ de tailles spécifiées. À cette fin, les fonctions analytiques définissant la surface, ainsi que leurs dérivées premières et éventuellement secondes, devront être fournies de manière externe (par exemple par un système de CAO). En outre, chaque carreau sera décrit à partir de son domaine de paramètres 2D. La méthode utilisée est basée sur une approche incrémentale de Delaunay adaptée à une métrique riemannienne.

Mots-clé : maillage surfacique, surface composée paramétrée, métrique riemannienne, triangulation de Delaunay, approche frontale.

Contents

1 Introduction

This introduction briefly presents the goal and the functionalities of the BLSURF software, and then provides some examples to illustrate its possibilities.

1.1 Presentation

Three-dimensional surface meshing is of the utmost importance in many numerical applications including the finite element method. It is a necessary step when one wants to construct the mesh of a solid object. A wide range of surfaces can be defined by means of composite parametric surfaces. Most of the surfaces are approximated by polynomial or rational parametric patches as is the case for most CAD-CAM modelers.

The method implemented in the BLSURF software is suitable for generating a mesh which conforms to given constraints (prescribed sizes of the elements in the vicinity of points of the surfaces) and closely approximates the geometry of the surface. It consists of meshing a 2D parametric domain, and a surface mesh is obtained when this is mapped to the 3D space. By discretizing first the interface curves which represent the common boundary of the patches, it has been adapted to surfaces made up of several parametric patches.

1.2 Data flow

As input, the surface must be defined by a set of parametric patches. The description of each patch consists of the domain of two parameters u and v, an analytical function $S(u, v)$ of class C^2, and also the first and second derivatives of $S(u, v)$. The parametric domains can be of arbitrary shapes. Validation tests have been carried out for different kinds of surfaces: spherical, toroidal, Bézier, B-spline, NURBS, CAD grids, etc. It is possible to specify the desired sizes of the elements (constant, dependent on curvature radii, or imposed by given functions).

As output, in version 0 of this software, an isotropic surface mesh is generated. Forthcoming versions will provide anisotropic meshes, curved quadratic elements and adaptive meshes (with size maps prescribed on background meshes).

1.3 A few examples

The following figures show three sample meshes created by the BLSURF software. Figure 1 shows a uniform mesh and a geometric mesh of the well-known Utah teapot, which is composed of 32 Bézier patches. Figure 2 shows the mesh of a Klein bottle defined by an analytic equation. This mesh conforms to both a given spiral metric and the geometry of the surface.

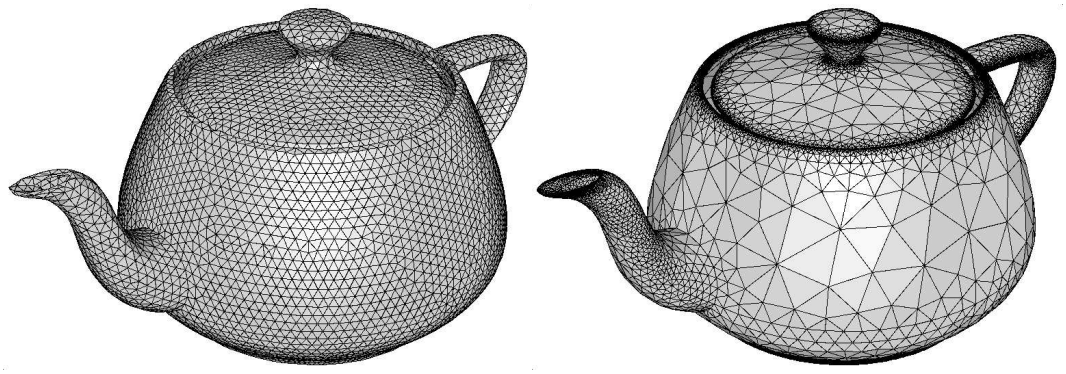


Figure 1: Utah teapot: uniform and geometric mesh.

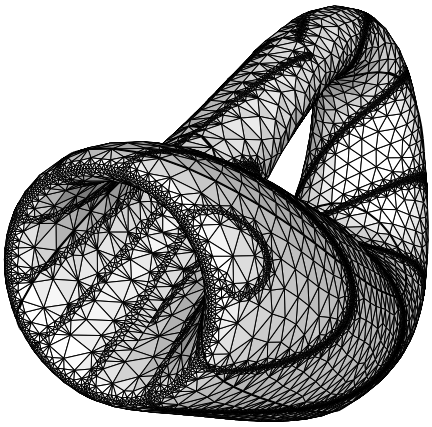


Figure 2: Klein bottle: both spiral and geometric mesh.

2 Modeling a composite surface

This section briefly presents the principle of modeling a single parametric patch, then a surface composed of several patches.

2.1 Parametric patch

Let us consider a parametric patch Σ in three-dimensional real space $\mathbb{R}^3$ (see figure 3 right). By definition, it represents the image of a parametric domain σ of $\mathbb{R}^2$ by a function $S(u,v)$:

$$\begin{bmatrix} u \\ v \end{bmatrix} \in \sigma \qquad \longmapsto \qquad S(u,v) = \begin{bmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{bmatrix} \in \Sigma \ .$$

The domain σ contains particular curved segments, for instance those making up the boundary. It may also contain predefined internal curved segments, for instance to model cracks. Each of these curved segments γ of $\mathbb{R}^2$ is the image by a function $C(t)$ of an interval $[a,b]$ of $\mathbb{R}$:

$$t \in [a,b] \qquad \longmapsto \qquad C(t) = \begin{bmatrix} u(t) \\ v(t) \end{bmatrix} \in \gamma \ .$$

Having defined the functions $S(u,v)$ and $C(t)$, the curved segment Γ of $\mathbb{R}^3$ is defined by a function $S(u(t),v(t))$, $t \in [a,b]$.

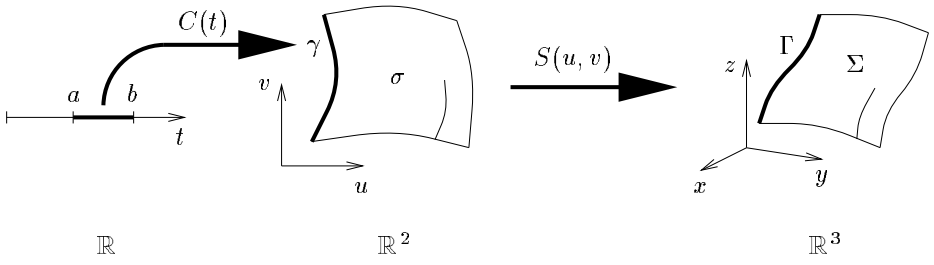


Figure 3: Simple parametric surface.

Please note: in what follows, to lighten the text and by abuse of language, we may use the term "curve" instead of "curved segment".

The preceding functions must satisfy certain properties:

($\mathcal{P}_0$) Functions $C(t)$ and $S(u,v)$ must be defined and continuous.

($\mathcal{P}_1$) Their first derivatives $\frac{dC}{dt}(t)$, $\frac{\partial S}{\partial u}(u,v)$ and $\frac{\partial S}{\partial v}(u,v)$ must also be defined and continuous. Moreover, the plane tangent to the surface must be defined everywhere, which is equivalent to: $\forall (u,v)$, $\frac{\partial S}{\partial u}(u,v) \neq 0$ and $\frac{\partial S}{\partial v}(u,v) \neq 0$, and these two vectors are not colinear.

($\mathcal{P}_2$) Their second derivatives $\frac{d^2C}{dt^2}(t)$, $\frac{\partial^2 S}{\partial u^2}(u,v)$, $\frac{\partial^2 S}{\partial u\,\partial v}(u,v)$ and $\frac{\partial^2 S}{\partial v^2}(u,v)$ must also be defined and continuous, in the case where the prescribed sizes of the mesh elements depend on curvature radii of surfaces and curves.

If property ($\mathcal{P}_1$) is not verified, the associated metrics are undefined at some points, and bad quality elements may appear. To remedy this problem, it is sometimes sufficient to give the values of the derivatives at a neighboring point (approximate solution). It is also possible to split a surface or a curve into several pieces, or else to modify its parameterization, as detailed in the examples below.

Example 1: Splitting a curve

Figure 4 shows the curve defined by:

$$u(t) = t, \qquad v(t) = |\sin(t)|, \qquad t \in [-\frac{\pi}{2}, +\frac{\pi}{2}]\ .$$

Because the first derivative of function $v(t)$ is discontinuous at $t = 0$, we may wish to split this curve into two pieces, $t \in [-\frac{\pi}{2}, 0]$ and $t \in [0, +\frac{\pi}{2}]$.

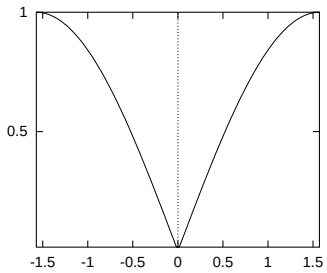


Figure 4: Curve $v(t) = |\sin(t)|$.

Example 2: Parameterization of a spherical surface

To parameterize a spherical surface, we can inversely define a projection from the 3D surface to the 2D parametric domain. In figure 5, the *orthogonal* projection of point P of the sphere gives point P_o, while the *stereographic* projection gives point P_s.

Let us consider the first projection method, called **orthogonal.** If $P = (x, y, z)$ is a point of the sphere with center $C = (0, 0, R)$ and radius R, its orthogonal projection is the point $P_o = (u, v)$ with $u = x$ and $v = y$. Inversely, a hemisphere can be parameterized by:

$$S_o(u,v) = \begin{bmatrix} x(u,v) = u \\ y(u,v) = v \\ z(u,v) = \sqrt{R^2 - u^2 - v^2} \end{bmatrix}.$$

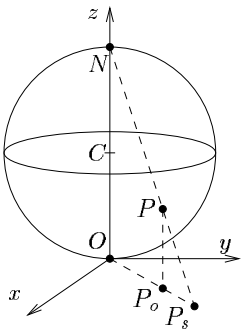


Figure 5: Orthogonal and stereographic projections.

The main drawback of this parameterization is that it is highly unstable near the "equator", for the limit $\frac{\partial z}{\partial u}(u,v)$ and $\frac{\partial z}{\partial v}(u,v)$ is infinity as $R^2 - u^2 - v^2$ goes to 0. As an illustration, figure 6 shows a uniform mesh on a plane disk (bottom) and the corresponding surface mesh (top left).

Another projection method, called **stereographic,** is used in cartography for polar regions. If $N = (0, 0, 2\,R)$ is the sphere's "North Pole", the stereographic projection of point P is defined as the intersection P_s of the straight line (NP) with the plane $z = 0$. We now have $P_s = (u, v)$ with $u = \frac{2\,R\,x}{2\,R - z}$ and $v = \frac{2\,R\,y}{2\,R - z}$. Conversely, the sphere without point N can be parameterized by:

$$S_s(u,v) = \frac{2\,R}{u^2 + v^2 + 4\,R^2} \begin{bmatrix} 2\,R\,u \\ 2\,R\,v \\ u^2 + v^2 \end{bmatrix}.$$

This latter parameterization is stable near the equator, continuously differentiable, and preserves angles (but not distances) [1]. As a consequence, a triangle which is equilateral on the parametric domain is also equilateral on the sphere, which leads to good quality elements for the surface mesh (see figure 6 top right).

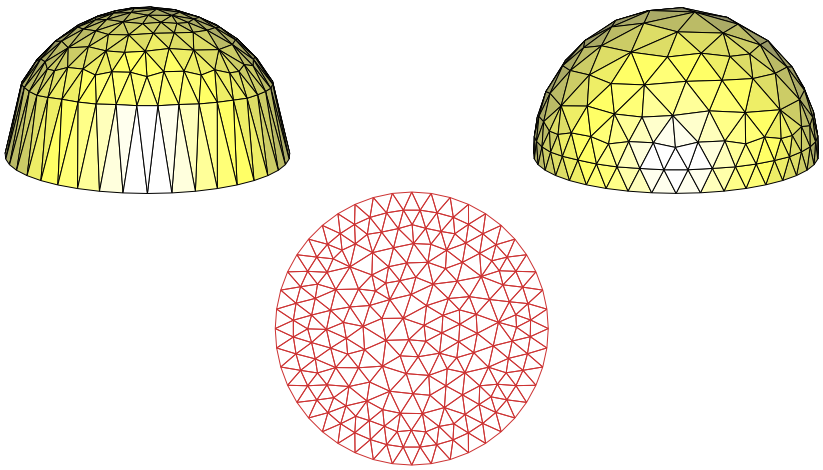


Figure 6: Bottom: uniform mesh of a plane disk. Top left: its inverse orthogonal projection. Top right: its inverse stereographic projection.

Example 3: Parameterizing a surface of revolution

A surface of revolution can be defined by a curve $C(t)$ in a revolving coordinate system XOY (see figure 7). Therefore, the equations of the curve and the surface are respectively:

$$C(t) = \begin{bmatrix} X(t) \\ Y(t) \end{bmatrix} \quad \text{and} \quad S(\varphi, t) = \begin{bmatrix} X(t)\ \cos\varphi \\ X(t)\ \sin\varphi \\ Y(t) \end{bmatrix} .$$

To see this more practically, let us take for definition intervals $t \in [0, 1]$ and $\varphi \in [0, \frac{\pi}{2}]$. Also, to show the influence of the parameterization method, let us suppose that $t = 0 \Rightarrow X(t) = 0$. Then, as φ varies from 0 to $\frac{\pi}{2}$ and $t = 0$, $S(\varphi, t)$ reduces to a point P of the Oz axis.

An initial method is to choose directly φ and t as parameters, their domain then being a rectangle. However, the inverse image of point P by the function $S(\varphi, t)$ is in that case a whole segment, the edge with ordinate $t = 0$. For all points (φ, t) of this edge, the function $S(\varphi, t)$ is constant as φ varies, and therefore the derivative $\frac{\partial S}{\partial \varphi}(\varphi, t)$ stays equal to 0 (this can also be easily found from the preceding equations).

A better approach consists in taking for parametric domain the quarter disk in the coordinate system (u, v) such that $u = t\ \cos\varphi$ and $v = t\ \sin\varphi$. The inverse image of point P then reduces to a single point, the center of the disk, which eliminates the previous problem.

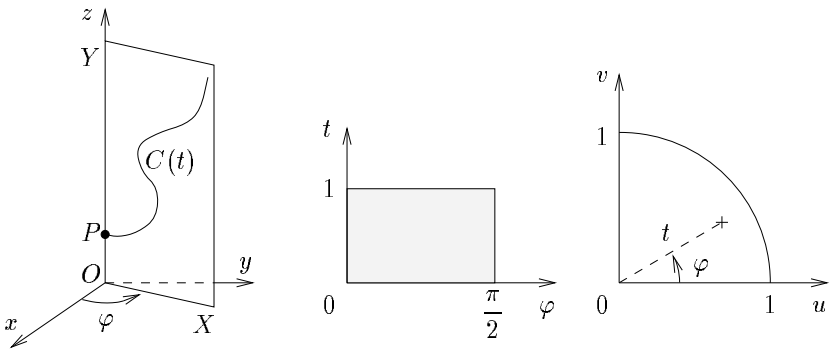


Figure 7: Surface of revolution.

Having studied the parameterization of one patch only, let us now consider the parameterization of a surface composed of several patches.

2.2 Composite surface

A composite surface is defined by a set of patches (see figure 8). Each patch is defined by a domain σ_i and a function $S_i(u,v)$.

The images in $\mathbb{R}^3$ of some curves of $\mathbb{R}^2$ can be geometrically identical. Generally, we also want them to be *topologically* identical, i.e. the surface mesh should be based on a unique discretization of these curves (called *interface curves*). This leads to *conformal* meshes (see in figure 9 examples of conformal and non conformal meshes). To this end, we identify the interface curves by giving them identical *references*. In the example in figure 8, the three-dimensional curve C_{10} is the image by $S_1(u,v)$ of curve c_{10} of domain σ_1 and the image by $S_3(u,v)$ of curve c_{10} of domain σ_3, hence the common reference which equals 10.

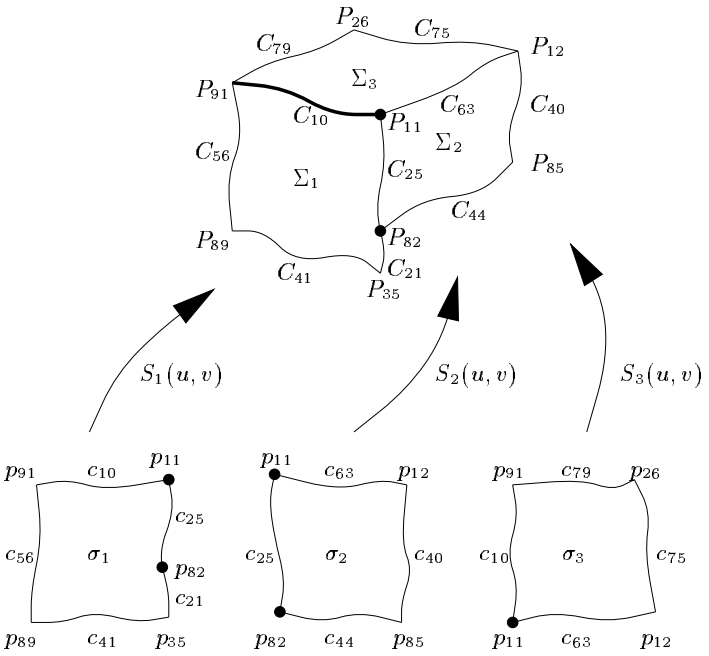


Figure 8: Composite parametric surface.

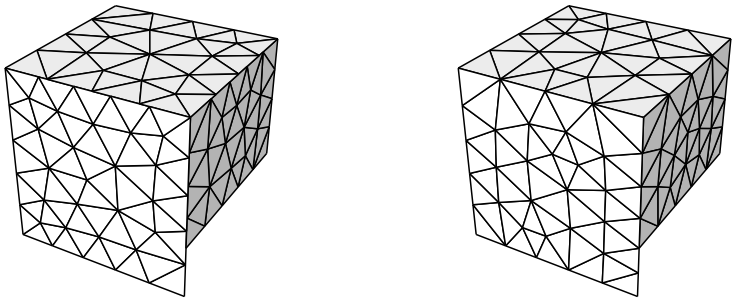


Figure 9: Non conformal mesh (left) and conformal mesh (right).

In the same manner, points having the same location in $\mathbb{R}^3$ are identified by identical references. Here, point P_{11} is the image of three different points p_{11}. Finally, to make patches Σ_1 and Σ_2 conformal, point p_{82} must appear in domain σ_1, though there is no discontinuity between curves c_{21} and c_{25} of this domain.

3 Input files and external functions

Based on the previous model, the BLSURF software needs information on the parameterization of surfaces and curves, and also on the parametric domains and their references. To this end, the software can call external functions gathered in a Fortran module named `cad_m`, or also read a data file `x.pardom`. Finally, a file called `blsurf.env` contains environment variables in order to control several functionalities of the software.

3.1 Module `cad_m`: external functions

The Fortran 90 module named `cad_m` contains the implementation of surfaces and curves parameterizations (mathematical functions $S(u, v)$ and $C(t)$ as well as their derivatives). Moreover, it contains optional functions to define the desired size of the elements. Naturally, all these functions must be compiled and linked to make the executable file of BLSURF (cf. section 5).

Figure 10 shows the general structure of this module. The most important functions are `surf0`, `surf1`, `surf2` which define the surfaces, and `curv0`, `curv1`, `curv2` which define the curves. The other functions are optional.

```
module cad_m
      ...     ! This part can contain global variables
contains
      subroutine cad_init
      ...     ! makes initializations, if necessary
      subroutine surf0(refs, uv, S)
      ...     ! returns S(u,v)
      subroutine surf1(refs, uv, Su, Sv)
      ...     ! returns the first derivatives of S(u,v)
      subroutine surf2(refs, uv, Suu, Suv, Svv)
      ...     ! returns the second derivatives of S(u,v)
      subroutine curv_int(refs, ic, a, b)
      ...     ! returns the definition interval of a function C(t)
      subroutine curv0(refs, ic, t, C)
      ...     ! returns C(t)
      subroutine curv1(refs, ic, t, Ct)
      ...     ! returns the first derivative of C(t)
      subroutine curv2(refs, ic, t, Ctt)
      ...     ! returns the second derivative of C(t)
      subroutine cad_hphys(refs, uv, h)
      ...     ! returns the desired size at a point of a surface
      subroutine cad_hphyc(refs, ic, t, h)
      ...     ! returns the desired size at a point of a curve
      subroutine cad_hphyp(refp, h)
      ...     ! returns the desired size at a point (extremity)
      subroutine cad_rads(refs, uv, rhos)
      ...     ! returns the radius of curvature of a surface
      subroutine cad_radc(refs, ic, t, rhoc)
      ...     ! returns the radius of curvature of a curve
end module cad_m
```

Figure 10: General structure of the Fortran 90 module `cad_m`.

We detail below each of the functions which make up this module (surfaces and curves parameterization, then sizes specification). At the end of this manual, a complete example of a module is given (cf. section 6.1).

3.1.1 Parameterization of surfaces

Let us recall that a single parameterized surface is mathematically defined by a function $S(u,v)$. The implementation of this function and its derivatives is realized by the four functions described below.

- `cad_init` is called only once, at the beginning of the execution of BLSURF, with the aim of making some initializations if necessary. Generally, the purpose is to read a CAD data file or the equivalent.

- `surf0(refs, uv, S)` implements a function $S(u,v)$. The *"intent(in)" dummy argument* `refs` *is the reference of a surface. The "intent(in)" dummy argument* `uv` *is an array of two coordinates (*$u =$ `uv(1)` *and* $v =$ `uv(2)`*). As a result, an array of three coordinates* $\texttt{S} = S(u,v)$ *is obtained.*

- `surf1(refs, uv, Su, Sv)` *gives, in a similar way, the first derivatives* $\texttt{Su} = \dfrac{\partial S}{\partial u}(u,v)$ *and* $\texttt{Sv} = \dfrac{\partial S}{\partial v}(u,v)$.

- `surf2(refs, uv, Suu, Suv, Svv)` *gives, in a similar way, the second derivatives* $\texttt{Suu} = \dfrac{\partial^2 S}{\partial u^2}(u,v)$, $\texttt{Suv} = \dfrac{\partial^2 S}{\partial u\,\partial v}(u,v)$ *and* $\texttt{Svv} = \dfrac{\partial^2 S}{\partial v^2}(u,v)$. *These second derivatives are really necessary only in the case where the desired sizes of the elements depend on the radii of curvature.*

The function `surf0`, which implements $S(u,v)$, can be fully programmed by the user, or simply call other functions in a CAD system. The two examples below show these two cases more precisely. Once the programmation of function `surf0` is completed, its derivatives `surf1` and `surf2` can be manually written, or else automatically obtained by a computer algebra system such as Maple [5] or Odyssée [6].

Example 1: full programmation

The Klein bottle represented in figure 2 is defined by the following parametric equations:

$$u, v \in [0, 2\,\pi] \qquad r = 4 - 2\,\cos u$$

$$x = \begin{cases} 6\,\cos u\,(1+\sin u) + r\,\cos u\,\cos v & \text{if } u \in [0, \pi[\\ 6\,\cos u\,(1+\sin u) - r\,\cos v & \text{if } u \in [\pi, 2\,\pi] \end{cases}$$

$$y = \begin{cases} 16\,\sin u + r\,\sin u\,\cos v & \text{if } u \in [0, \pi[\\ 16\,\sin u & \text{if } u \in [\pi, 2\,\pi] \end{cases}$$

$$z = r\,\sin v$$

Because of the periodicity of this surface, the parametric domain $[0, 2\,\pi]^2$ is split into four squares. This gives the following `surf0` function:

```
subroutine surf0(refs, uv, S)
  integer :: refs
  double precision :: uv(2), S(3), u, v, cosu, sinu, cosv, sinv, r

  u = uv(1) ; v = uv(2)
  cosu = cos(u) ; sinu = sin(u)
  cosv = cos(v) ; sinv = sin(v)
  r = 4 - 2*cosu
  if (refs <= 2) then
     S(1) = 6*cosu*(1+sinu) + r*cosu*cosv   ! x (case 1)
     S(2) = 16*sinu + r*sinu*cosv           ! y (case 1)
  else
     S(1) = 6*cosu*(1+sinu) - r*cosv        ! x (case 2)
     S(2) = 16*sinu                         ! y (case 2)
  end if
  S(3) = r*sinv                             ! z
end subroutine surf0
```

All the possibilities of the Fortran 90 language can be used, such as `select case` for multi-way branches according to `refs` or the use of arrays indexed by `refs`.

Example 2: call to a CAD system

The case of a call to a CAD system is simply of the form:

```
subroutine surf0(refs, uv, S)
  integer :: refs
  double precision :: uv(2), S(3)

  call ...  ! call to one or several functions of the CAD system
end subroutine surf0
```

This method theoretically makes it possible to adapt to any CAD system. However, this involves an extra software layer, which should be eliminated to save CPU time.

3.1.2 Parameterization of curves

Let us recall that a curve is mathematically defined by a function $C(t), t \in [a, b]$. The implementation of this function and its derivatives is realized by the four functions described below.

- `curv_int(refs, ic, a, b)` gives the bounds of interval $[a, b]$. We will see later that file `x.pardom` contains, for each surface, its reference `refs` and the descriptions of the `nc` curves which belong to it (cf. section 3.2). The "*intent(in)" dummy arguments of* `curv_int`*, that are* `refs` *and* `ic` *(*$1 \leq$ `ic` $\leq$ `nc`*), refer to such a curve.*

- `curv0(refs, ic, t, C)` *implements a function* $C(t)$*. The "intent(in)" dummy arguments* `refs` *and* `ic` *refer to such a curve as previously. The third "intent(in)" dummy argument* `t` *is a scalar value that lies between the bounds* `a` *and* `b` *returned by* `curv_int`*. As a result, an array of two coordinates* `C` $= C(t)$ *is obtained.*

- `curv1(refs, ic, t, Ct)` *gives, in a similar way, the first derivative* `Ct` $= \frac{d}{dt}C(t)$*.*

- `curv2(refs, ic, t, Ctt)` *gives, in a similar way, the second derivative* `Ctt` $= \frac{d^2}{dt^2}C(t)$*. This second derivative is really necessary only in the case where the desired sizes of the elements depend on the radii of curvature.*

The remarks mentioned in the previous section concerning surfaces (full programmation, call to a CAD system, and automatic differentiation) also apply to curves.

3.1.3 Specification of sizes

In most simple cases, no environment variable is specified and no optional function is programmed. Then, a uniform mesh is obtained, the size of the elements being $diag/50$, where $diag$ represents the length of the diagonal of the bounding box (cf. examples in section 6). This size can be modified by the environment variable `hphydef`.

In other cases, the BLSURF software provides many possibilities that are detailed here. To obtain the size h prescribed at a point P of the surface to be meshed, BLSURF uses in fact a **physical** size h_{phy} (chosen by the user) and a **geometric** size h_{geo} (intended to conform to the geometry of the surface to be meshed, considering the curvatures). The calculation of these sizes h_{phy} and h_{geo} is detailed in paragraphs (a) and (b) of this section.

All the physical sizes h_{phy} are "trimmed", i.e. each size is forced between two bounds $h_{phy.min}$ and $h_{phy.max}$. More precisely, if $h_{phy} < h_{phy.min}$ then $h_{phy} = h_{phy.min}$; else if $h_{phy} > h_{phy.max}$ then $h_{phy} = h_{phy.max}$. In a similar way, all the geometric sizes h_{geo} are trimmed between two bounds $h_{geo.min}$ and $h_{geo.max}$. In practice, these bounds are given by environment variables `hphymin`, `hphymax`, `hgeomin` and `hgeomax` (cf. section 3.3). By default, both lower bounds have a value equal to $diag/500$ and both upper bounds have a value equal to $diag/5$, where $diag$ represents the length of the diagonal of the box bounding the surface to mesh.

Finally, the prescribed size h is obtained in the following way:

- Compute the physical size h_{phy} and trim it.
- Compute the geometrical size h_{geo} and trim it.
- The prescribed size is $h = \min(h_{phy}, h_{geo})$.

It is also possible to impose only a physical size while ignoring the geometrical size (giving finally $h = h_{phy}$), or conversely ($h = h_{geo}$).

Let us now consider more precisely how the physical size and the geometrical size at a given point P are computed.

(a) Computation of the physical size

This computation depends on the chosen option, which is governed by the environment variable `hphy_flag`. The value of this variable can be 0, 1 (by default) or 2.

If `hphy_flag` $= 0$, the physical size h_{phy} is ignored. In this case, we must have `hgeo_flag` $\neq 0$, finally giving $h = h_{geo}$.

If `hphy_flag` $= 1$, the size is given by the value of the environment variable `hphydef`.

If `hphy_flag` $= 2$, it is obtained by querying functions `cad_hphys` (surfaces), `cad_hphyc` (curves) and `cad_hphyp` (points). Each function can either return a value `h` (which is then trimmed between the two bounds `hphymin` and `hphymax`), or "not answer" (by not assigning a value to `h`), thus providing great flexibility in the specification of the sizes. The computation depends whether point P is internal to a surface, internal to a curve, or at the end of several curves:

- If point P is internal to a **surface,** `cad_hphys` is queried. If it does not answer, one interpolates with the values at the vertices of the discretized interface curves.
- If point P is internal to a **curve,** `cad_hphyc` is queried first. If it does not answer, `cad_hphys` is queried for every adjacent surface and the mean of the returned values is computed. If there is no answer, sizes h_1 and h_2 at both ends of the curve are considered (see next item) and the interpolated value is computed.
- If point P is at the **extremity** of several curves, `cad_hphyp` is queried first. If it does not answer, `cad_hphyc` is queried for every adjacent curve and the mean of the returned values is computed. If there is no answer, `cad_hphys` is queried for every adjacent surface and the mean of the returned values is computed. If there is no answer again, the default value `hphydef` is retained.

In the above paragraph (if `hphy_flag` $= 2$), in order to compute the mean of several values, the arithmetic mean is used by default, but this can be modified by the environment variable `hmean_flag`. In the same way, in order to interpolate two values, a linear interpolation is used by default, but this can be modified by `hinterpol_flag` (cf. section 3.3).

Therefore, when `hphy_flag` = 2, the three functions `cad_hphys`, `cad_hphyc` and `cad_hphyp` can be programmed:

- `cad_hphys(refs, uv, h)` returns (or not) a prescribed size `h` at a point of surface `refs` with coordinates `uv` in the parametric domain.
- `cad_hphyc(refs, ic, t, h)` returns (or not) a prescribed size `h` at a point of curve (`refs`, `ic`) with parameter `t`.
- `cad_hphyp(refp, h)` returns (or not) a prescribed size `h` at an extremity of reference `refp`.

Let us recall that each function can either return a value `h` (which is then trimmed between two bounds `hphymin` and `hphymax`), or "not answer" (by not assigning a value to `h`, cf. examples in section 6). This is because the internal call from BLSURF software is of the form :

```
h = hempty              ! hempty is a "very negative" constant, e.g. -1.d38
call cad_hphys(..., h)  ! call cad_hphys,   call cad_hphyc   or   call cad_hphyp
if (h /= hempty) then
   h = ...   ! trim between hphymin and hphymax
else
   h = ...   ! compute another value
end if
```

(b) Computation of the geometric size

The computation of the geometric size at a given point P depends on the chosen option, which is governed by the environment variable `hgeo_flag`. The value of this variable can be 0, 1 (by default) or 2.

If `hgeo_flag` = 0, the geometric size h_{geo} is ignored. In this case, we must have `hphy_flag` $\neq 0$, finally giving $h = h_{phy}$.

If `hgeo_flag` = 1, it is computed by the BLSURF software in the following way:

- If point P is internal to a **surface,** $h_s = \lambda\ \rho_s$ is computed, where λ is a coefficient and ρ_s is the radius of curvature of the surface.
- If point P is internal to a **curve,** BLSURF computes the smallest size h_s induced by the adjacent surfaces, and the size $h_c = \lambda\ \rho_c$, where ρ_c is the radius of curvature proper to the curve. Finally, $h_{geo} = \min(h_s, h_c)$ is retained.

- If point P is at the **extremity** of several curves, BLSURF computes the smallest size h_s induced by the adjacent curves,

The coefficient λ is computed in such a way that a given tolerance angle (defined by the environment variable `angle_mesh`) is respected.

If `hgeo_flag` = 2, the computations are similar but the radii of curvature are not computed by BLSURF: they are given by the external functions `cad_rads` for surfaces and `cad_radc` for curves. The purpose is to optimize these computations in certain particular cases (spheres or cylinders for instance) since BLSURF usually makes a more general computation. The description of these two functions is the following:

- `cad_rads(refs, uv, rhos)` returns the radius of curvature `rhos` of surface `refs` at the point with coordinates `uv` in the parametric domain.
- `cad_radc(refs, ic, t, rhoc)` returns the radius of curvature `rhos` of curve (`refs`, `ic`) at the point with parameter `t`.

In the case of an infinite radius (plane surface or straight line), these functions must return a "big value", for instance `huge(1.)` in Fortran 90.

3.2 Input file `x.pardom`: parametric domains

The file `x.pardom` serves to complement the above descriptions of the different parametric domains, principally by giving the references of curves and points. It is no longer a Fortran program but a simple text file (ASCII).

The general form of this file is a nested structure as is shown in figure 11. It contains an integer number `ns`, followed by `ns` blocks. Each block contains the integers `refs`, `orientation`, `nc`, followed by `nc` blocks, then the integer `np` followed by `np` blocks.

The different elements contained in this file are detailed below. Some complete examples are given later (cf. section 6).

- `ns`: number of surfaces (or patches).
- `refs`: reference of the surface.
- `orientation`: integer whose *absolute value* gives a curve number `ic` and whose *sign* determines the orientation of the surface: if the sign is positive, the surface lies on the left of curve `ic` ; if it is negative, the surface lies on the right. The type of the curve `ic` must be "boundary" (`typc` = 1).

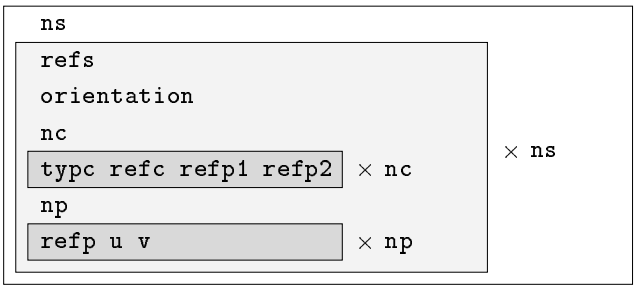


Figure 11: General structure of the text file `x.pardom`.

- `nc`: number of curves.
- `typc`: type of the curve: 1 "boundary", 2 "internal". A curve of type 1 is a part of the boundary of the sub-domain. A curve of type 2 is internal to a sub-domain, i.e. the edges of the triangles of the final mesh will contain its discretization. Examples of internal curves are given in section 6.2.
- `refc`: reference of the curve (to identify identical curves in 3D space).
- `refp1 refp2`: references of extremities 1 and 2 (obtained respectively for $t = a$ and $t = b$).
- `np`: number of internal points.
- `refp`: reference of the internal point.
- `u v`: coordinates of the internal point in the parametric domain.

Remarks

The name of the file is by default `x.pardom`, but its prefix can be modified by the environment variable `pref`.

If the reference of the very first curve is zero (`is` = `ic` = 1 ⇒ `refc` = 0), the BLSURF software automatically gives references by searching points that have the same location or are neighboring (cf. environment variable `eps_glue`). The results are saved in a file named `x.pardom.new`, which can be reused afterwards.

Two distinct surfaces must have different references (`is1` ≠ `is2` ⇒ `refs1` ≠ `refs2`). This reference is used in particular by functions of module `cad_m` (cf. section

3.1). As a practical application, it is possible to mesh only one patch by giving `ns` = 1, followed by the description of the patch containing its reference.

If several curves have the same reference, it is the definition of the **first curve** (in order of appearance in file `x.pardom`) which is used by the mesh generator.

For a given patch, all the boundary references must be different. For example, to represent a cylinder, at least two patches are necessary because of the periodicity of this surface.

References `refp1` and `refp2` at the extremities of one curve `refc` **must not** be identical. This is because they determine the direction of the curve.

After file `x.pardom` is read, the BLSURF software checks if the preceding properties hold.

3.3 Input file `blsurf.env`: environment variables

The BLSURF software is governed by a set of environment variables. Each variable has a predefined name and a default value. File `blsurf.env` makes it possible to modify the values of these variables. Once again, it is an ASCII text file.

The general structure of this file is illustrated by figure 12. Several successive actions are invoked by the keyword `call`. The BLSURF software executes a loop where it initializes the environment variables by their default values, reads their new values until the keyword `call` is reached, runs the corresponding action, re-initializes the environment variables, re-reads the new values, and so forth until `call exit` is reached. Some complete examples are given in section 6.

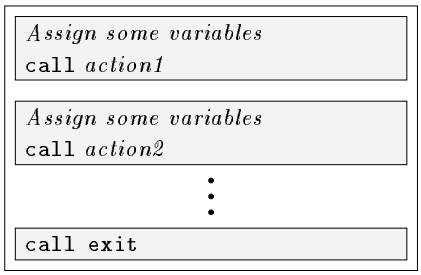


Figure 12: General structure of the text file `blsurf.env`.

◊	Name	Type	Default value
	`CheckAdjacentEdges`	integer	0
	`CheckCloseEdges`	integer	0
	`CheckWellDefined`	integer	0
	`CoefRectangle`	real	0.25
	`LSS`	real	0.5
	`angle_mesh`	real	8.
	`angle_smo`	real	1.
◊	`call`	characters	`"exit"`
	`element`	characters	`"p1"`
	`eps_collapse`	real	0.
	`eps_ends`	real	$diag/500$
	`eps_glue`	real	$diag/500$
	`flag`	integer	0
	`format`	characters	`"mesh"`
◊	`frontal`	integer	1
◊	`hgeo_flag`	integer	0
	`hgeomax`	real	$diag/5$
	`hgeomin`	real	$diag/500$
	`hinterpol_flag`	integer	0
	`hmean_flag`	integer	0
◊	`hphy_flag`	integer	1
	`hphydef`	real	$diag/50$
	`hphymax`	real	$diag/5$
	`hphymin`	real	$diag/500$
	`option`	characters	`""`
	`pref`	characters	`"x"`
	`refs`	integer	1
◊	`verb`	integer	10

Figure 13: List of the environment variables in file `blsurf.env`.

The table in figure 13 gives an alphabetical list of all the environment variables. These are rather numerous, but in general only a few of them are necessary. They are indicated in the figure by a diamond-shaped sign ($\diamondsuit$). As previously, *diag* represents the length of the diagonal of the box which bounds the surface to mesh. We detail these variables below, giving for each its name, its default value between brackets [ ] and its description.

- `CheckAdjacentEdges` [0], `CheckCloseEdges` [0] and `CheckWellDefined` [0]. The BLSURF software calls n_1 times the subroutine `CheckCloseEdges`, n_2 times `CheckAdjacentEdges` and n_3 times `CheckWellDefined`. The purpose of these subroutines is to improve the mesh of domains having narrow parts [8]: at each iteration, `CheckCloseEdges` decreases the sizes of the edges when two boundary curves are neighboring (see figure 14), `CheckAdjacentEdges` balances the sizes of adjacent edges, and `CheckWellDefined` checks if the parametric domain is well defined. The three environment variables having the same names as these subroutines represent the respective numbers of iterations n_1, n_2 and n_3 (by default, 0 for each).

- `CoefRectangle` [0.25]. This variable defines the relative thickness of the rectangles used by subroutine `CheckCloseEdges` (see above).

- `LSS` [0.5]. LSS is an abbreviation for "length of sub-segment". From the specifications of the sizes, a Riemannian metric $\mathcal{M}$ is defined in such a way that the edges of the desired mesh are of length 1 in this metric. To compute the length $L_{\mathcal{M}}$ of an edge, the approximate value of an integral is used. If the value obtained is less than `LSS`, the result is considered to be correct. Otherwise, the edge is recursively split into two sub-edges. As a practical consequence, the

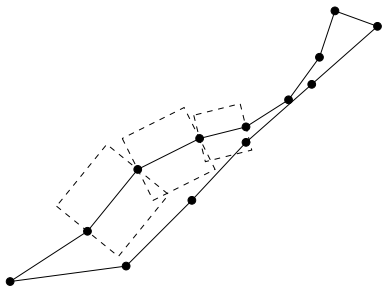


Figure 14: Checking close edges (subroutine `CheckCloseEdges`).

smaller `LSS` is, the more accurate the computations are, but the CPU time and the necessary memory increase.

- `angle_mesh` [8.]. This variable defines an angle θ (in degrees) which represents the tolerance of a geometric mesh, both for surfaces and curves. In the case of a surface mesh, θ is the limiting angle between the plane of a triangle of the mesh and each of the tangent planes at the three vertices. In the case of a curve discretization, θ is the limiting angle between an edge of the discretization and the tangent to the curve. Clearly, the smaller the angle is, the closer the mesh is to the exact surface.

- `angle_smo` [1.]. This variable defines an angle θ (in degrees) which represents the tolerance used for the geometric support of the curves (called *smooth*). This support approximates curves with polygonal segments. Angle θ is the limiting angle between an edge of the polygonal segment and the tangent to the curve. To ensure the inequality `angle_smo` $\leq$ `angle_mesh`, BLSURF can force the value of `angle_smo`.

- `call` [`"exit"`] activates a predefined action of BLSURF, and resets the environment variables to their default values. The actions that are admitted at present are: `"inimesh"` $\Rightarrow$ create a mesh (which is the initial mesh in the case of an adaptive process), `"export"` $\Rightarrow$ export a mesh into a file, and `"exit"` $\Rightarrow$ exit from the program.

- `element` [`"p1"`]. This variable must not be modified in version 0 of the BLSURF software. It determines the type of the elements of the mesh to be created.

- `eps_collapse` [0.]. If `eps_collapse` > 0, BLSURF removes curves whose lengths are less than `eps_collapse`. Here, to obtain an approximate value of the length of a curve, the latter is arbitrarily split into 20 edges.

- `eps_ends` [$diag/500$]. This variable is used to detect curves whose lengths are very small, which sometimes constitutes an error. A message is printed if $\|P_2 - P_1\| <$ `eps_ends`, where P_1 and P_2 are the extremities of a curve.

- `eps_glue` [$diag/500$]. Optionally, the BLSURF software automatically gives references by searching for points that have the same location or are neighboring (cf. section 3.2). In this case, two points P_1 and P_2 are considered to be neighboring if $\|P_2 - P_1\| <$ `eps_glue`.

- `flag` [0]. This variable is used to modify the contents of an exported mesh file (cf. section 4). At present, the possible values are 0 or 1.

- `format` ["mesh"]. This variable must not be modified in version 0 of the BL-SURF software. It determines the format of the file to export (cf. section 4).

- `frontal` [1]. If `frontal` = 1, the mesh generator inserts points with an advancing front method. If `frontal` = 0, it inserts them with an algebraic method (on internal edges). This latter method is faster but generates less regular meshes.

- `hgeo_flag` [0]. Option for computing the geometric size (cf. section 3.1.3). The possible values are 0, 1 or 2.

- `hgeomax` [$diag/5$]. Maximum geometric size (cf. section 3.1.3).

- `hgeomin` [$diag/500$]. Minimum geometric size (cf. section 3.1.3).

- `hinterpol_flag` [0] determines the computation of an interpolated value v between two points P_1 and P_2 on a curve. Let h_1 be the value at point P_1, h_2 be the value at point P_2, and t be a parameter varying from 0 to 1 when moving from P_1 to P_2. If `hinterpol_flag` = 0, the interpolation is linear: $v = h_1 + t\,(h_2 - h_1)$. If `hinterpol_flag` = 1, it is geometric: $v = h_1 \left(\frac{h_2}{h_1}\right)^t$. If `hinterpol_flag` = 2, it is sinusoidal: $v = \frac{h1 + h2}{2} + \frac{h1 - h2}{2} \cos(\pi\, t)$. See also section 3.1.3(a) and figure 23.

- `hmean_flag` [0] determines the computation of the mean m of n values h_i. If `hmean_flag` = -1, the minimum $m = \min_n h_i$ is computed. If `hmean_flag` = 0 or 2, the arithmetic mean $m = \left(\sum_n h_i\right)/n$ is computed. If `hmean_flag` = 1, the geometric mean $m = \left(\prod_n h_i\right)^{1/n}$ is computed. See also section 3.1.3(a) and figure 22.

- `hphy_flag` [1]. Option for computing the physical size (cf. section 3.1.3). The possible values are 0, 1 or 2.

- `hphydef` [$diag/50$]. Predefined physical size. Note that this size is trimmed into the interval [`hphymin`, `hphymax`].

- `hphymax` [$diag/5$]. Maximum physical size (cf. section 3.1.3).

- `hphymin` [$diag/500$]. Minimum physical size (cf. section 3.1.3).
- `option` [""]. Option used when exporting files (cf. section 4). Possible values are "", `"allsurf3d"`, `"allsurf2d"`, `"onesurf3d"`, `"onesurf2d"` and `"smo3d"`.
- `pref` [`"x"`]. Prefix of the files generated by BLSURF.
- `refs` [1]. reference of a surface, used when exporting files (cf. section 4).
- `verb` [10] defines the percentage of "verbosity" of the program. This value is an integer and must be between 0 (no printing, at least in theory) and 100 (maximum printing).

Syntax of a `blsurf.env` file. To specify briefly the syntax of an environment file, it is made up of a sequence of couples of words, where each couple represents the **name** and the **value** of a variable. It is possible to include comments, which as in Fortran 90 start with an exclamation mark (!) and finish at the end of the line. The words can be separated by one or several "blank" characters. A "blank" character can be a space, a tabulation or an equal sign (=). Examples are provided in section 6.

4 Output files (exported meshes)

As indicated in section 3.3, the line `call export` causes the exportation of a mesh into a file. Generally, it is the mesh of the complete surface, in a rather simple format (list of points and triangles). We give here some details about this mesh exportation.

When `call export` is activated, the software considers the environment variable `option`.

If `option = ""` or `"allsurf3d"`, it exports the 3D mesh of the whole surface.

If `option = "allsurf2d"`, it exports into a unique file all the meshes of the parametric domains (2D). This can only be interesting when they don't overlap (see also `"onesurf2d"`).

If `option = "onesurf3d"`, it exports the 3D mesh of only one patch, whose reference is given by the environment variable `refs`. To obtain several mesh files, just invoke `call export` several times.

If `option = "onesurf2d"`, it exports in the same way the 2D mesh of only one patch.

If `option = "smo3d"`, it exports the geometric support of the curves (so-called *smooth*).

The environment variable `flag` is used to modify the contents of the output file. If `flag` = 0, one obtains a classical surface mesh made of a list of vertices and triangles. If `flag` = 1, one also obtains the list of the edges involved in the curves discretization (see figure 17).

The name of the exported file is by default `x.mesh`, but its prefix can be modified by the environment variable `pref`.

Now, all the output files are in the `mesh` format, which is very general (see reference [7], section 10.3, *a general data structure*). Such a file is made up of a list of fields, and we specify below those used by the BLSURF software:

`MeshVersionFormatted 0` indicates the release identificator and the type of file.

`Dimension` is followed by 2 or 3 (2D or 3D).

`Vertices` is followed by the number of vertices, then each vertex is of the form: `u v refp` in 2D or `x y z refp` in 3D. The reference `refp` of the point is the one given in the input file `x.pardom`, or 0 if the point has been created by the mesh generator.

`Triangles` is followed by the number of triangles, then each triangle is given in the form `p1 p2 p3 refs` (indices of the previously given vertices and reference of the surface).

`Edges` is followed by the number edges of discretized curves, then each edge is given in the form `p1 p2 refc` (indices of the previously given vertices and reference of the curve).

`End` indicates the end of the file.

To illustrate these specifications, we give below a few examples of files in the `mesh` format (meshes of 2D and 3D surfaces, and discretizations of 3D curves).

2D surfaces:

```
MeshVersionFormatted 0
Dimension
2
Vertices
9
-2.5000000  -2.5000000   1
 2.5000000  -2.5000000   2
 2.5000000   2.5000000   4
-2.5000000   2.5000000   6
 0.0000000  -2.5000000   0
 2.5000000   0.0000000   0
 0.0000000   2.5000000   0
-2.5000000   0.0000000   0
 0.0000000   0.0000000   1
Triangles
8
7 4 8 1
6 9 5 1
9 8 5 1
1 5 8 1
7 8 9 1
3 7 6 1
6 5 2 1
6 7 9 1
End
```

3D surfaces:

```
MeshVersionFormatted 0
Dimension
3
Vertices
9
-2.5000000  -2.5000000   6.2500005   1
 2.5000000  -2.5000000   6.2500000   2
 2.5000000   2.5000000   6.2499995   4
-2.5000000   2.5000000   6.2500000   6
 0.0000000  -2.5000000   3.1250002   0
 2.5000000   0.0000000   3.1249998   0
 0.0000000   2.5000000   3.1249998   0
-2.5000000   0.0000000   3.1250002   0
 0.0000000   0.0000000   0.0000000   1
Triangles
8
7 4 8 1
6 9 5 1
9 8 5 1
1 5 8 1
7 8 9 1
3 7 6 1
6 5 2 1
6 7 9 1
End
```

3D curves:

```
MeshVersionFormatted 0
Dimension
3
Vertices
33
-2.5000000  -2.5000000   6.2500000   1
...
Edges
29
1 2   1
2 3   1
...
End
```

5 Installing the software

The BLSURF software has several source files:

- a very short main program, written in Fortran 90, named `blsurf.f90`.

```
program blsurf
  use  blsurf_m
  call blsurf_s
end program blsurf
```

- Fortran modules: `blsurf_m.f90`, `blmc_m.f90`, `share_m.f90` and `sharef_m.f` (the last one being in the old fixed format, hence a different suffix `.f`).
- a C program: `bltms_m.c` (bidimensional mesh generator).
- a sample module `cad_m.f90`.

Generally, it is necessary to rewrite the module `cad_m.f90`, which essentially contains the implementation of the parameterization of surfaces and curves. This part can be fully programmed by the user, or reduce to a simple call to a CAD system.

The preceding source files must be compiled to create the corresponding object files: `blsurf.o`, `blsurf_m.o`, `blmc_m.o`, `share_m.o`, `sharef_m.o`, `bltms_m.o` and `cad_m.o`.

If the software is autonomous, all these object files must be linked together (by a link editor) to make the executable file `blsurf`. The user will create the two ASCII input files, `x.pardom` and `blsurf.env` (cf. sections 3.2 and 3.3). He/she will then exploit the output files (cf. section 4), using for instance a personal vizualisation tool.

However, the software can also be integrated into an existing CAD system. In this case, all the previous object files, except `blsurf.o` which corresponds to a main program, should be linked to those which make up this system. This time, the CAD system itself could also generate the files `x.pardom` and `blsurf.env`, or read the output files of BLSURF, in a transparent way for the users. Finally, inputs and outputs of files can be replaced by associated arguments, in order to improve the performances.

6 Examples of use

In this section, we present in a detailed way the simple example of a paraboloid. Then, we will tackle the well known example of the Utah teapot (32 Bézier patches) and the bust of Victor Hugo (discrete cylindrical surface).

6.1 Paraboloid

Let us consider the part of a paraboloid defined by:

$$\begin{bmatrix} u \\ v \end{bmatrix} \in [-2.5,\ +2.5]^2 \qquad \longmapsto \qquad S(u,v) = \begin{bmatrix} x(u,v) = u \\ y(u,v) = v \\ z(u,v) = \dfrac{u^2+v^2}{2} \end{bmatrix}.$$

Each edge of the square $[-2.5,\ +2.5]^2$ has an equation like $C(t) = P + t\ \overrightarrow{PQ}$, $t \in [0,1]$, where P and Q are the extremities of the edge. The corresponding implementation of surfaces and curves is entirely given below:

```
module cad_m     ! PARABOLOID

  double precision :: segments(4, 4)

contains

  subroutine cad_init
    segments(1,:) = (/ -2.5, -2.5, +2.5, -2.5 /)
    segments(2,:) = (/ +2.5, -2.5, +2.5, +2.5 /)
    segments(3,:) = (/ +2.5, +2.5, -2.5, +2.5 /)
    segments(4,:) = (/ -2.5, +2.5, -2.5, -2.5 /)
  end subroutine cad_init

  subroutine surf0(refs, uv, S)
    integer :: refs
    double precision :: uv(2), S(3), u, v
    u = uv(1) ; v = uv(2)
    S(1) = u
    S(2) = v
    S(3) = (u**2+v**2) / 2
  end subroutine surf0
```

```
subroutine surf1(refs, uv, Su, Sv)
  integer :: refs
  double precision :: uv(2), Su(3), Sv(3), u, v
  u = uv(1) ; v = uv(2)
  Su(1) = 1 ; Su(2) = 0 ; Su(3) = u
  Sv(1) = 0 ; Sv(2) = 1 ; Sv(3) = v
end subroutine surf1

subroutine surf2(refs, uv, Suu, Suv, Svv)
  integer :: refs
  double precision :: uv(2), Suu(3), Suv(3), Svv(3)
  Suu(1) = 0 ; Suu(2) = 0 ; Suu(3) = 1
  Suv(1) = 0 ; Suv(2) = 0 ; Suv(3) = 0
  Svv(1) = 0 ; Svv(2) = 0 ; Svv(3) = 1
end subroutine surf2

subroutine curv_int(refs, ic, a, b)
  integer :: refs, ic
  double precision :: a, b
  a = 0.d0 ; b = 1.d0
end subroutine curv_int

subroutine curv0(refs, ic, t, C)
  integer :: refs, ic
  double precision :: t, C(2)
  C(1) = segments(ic,1) + t * (segments(ic,3) - segments(ic,1))
  C(2) = segments(ic,2) + t * (segments(ic,4) - segments(ic,2))
end subroutine curv0

subroutine curv1(refs, ic, t, Ct)
  integer :: refs, ic
  double precision :: t, Ct(2)
  Ct(1) = segments(ic,3) - segments(ic,1)
  Ct(2) = segments(ic,4) - segments(ic,2)
end subroutine curv1

subroutine curv2(refs, ic, t, Ctt)
  integer :: refs, ic
  double precision :: t, Ctt(2)
  Ctt(1) = 0.d0
  Ctt(2) = 0.d0
end subroutine curv2
```

```
  subroutine cad_hphys(refs, uv, h)
    integer :: refs
    double precision :: uv(2), h
    ! here, the value of h is not assigned to
  end subroutine cad_hphys

  subroutine cad_hphyc(refs, ic, t, h)
    integer :: refs, ic
    double precision :: t, h
    ! here, the value of h is not assigned to
  end subroutine cad_hphyc

  subroutine cad_hphyp(refp, h)
    integer :: refp
    double precision :: h
    ! here, the value of h is not assigned to
  end subroutine cad_hphyp

  subroutine cad_rads(refs, uv, rhos)
    integer :: refs
    double precision :: uv(2), rhos
    print *, "cad_rads should not be called", refs, uv, rhos
    stop
  end subroutine cad_rads

  subroutine cad_radc(refs, ic, t, rhoc)
    integer :: refs, ic
    double precision :: t, rhoc
    print *, "cad_radc should not be called", refs, ic, t, rhoc
    stop
  end subroutine cad_radc

end module cad_m
```

The file `x.pardom`, which describes the parametric domains, is first given **without** the references of curves and points. This has the effect of generating automatically the file `x.pardom.new`:

File `x.pardom`:

```
1   ! ns (number of surfaces)

1   ! refs (reference of the surface)
1   ! orientation
4   ! nc (number of curves)
1 0 0 0   ! typc refc refp1 refp2
1 0 0 0
1 0 0 0
1 0 0 0
0   ! np (number of internal points)
```

File `x.pardom.new`:

```
1   ! ns (number of surfaces)

1   ! refs (reference of the surface)
1   ! orientation
4   ! nc (number of curves)
1 1 1 2   ! typc refc refp1 refp2
1 2 2 4
1 3 4 6
1 4 6 1
0   ! np (number of internal points)
```

At first, file `blsurf.env` calls the three base functions of BLSURF and keeps all the default values:

File `blsurf.env`:

```
call inimesh
call export
call exit
```

Consequently, the exported file has the name `x.mesh` and contains a uniform mesh, obtained with an advancing front method. The length of the diagonal of the box bounding is $diag = 7.7$, hence the default physical size `hphydef` $= diag/50 = 0.154$. Figure 15 represents this mesh, which contains 2599 verticess and 4980 triangles.

A coarser uniform mesh is obtained by modifying files `x.pardom` and `blsurf.env` (see below). Here, we impose an internal point at the center of the square, we prescribe a size `hphydef` $= 1.5$ (and increase `hphymax` to the same value) and we export both the 2D mesh of the parametric domain and the 3D mesh of the surface. Figure 16 shows these 2D and 3D meshes, where each contains 45 vertices and 64 triangles.

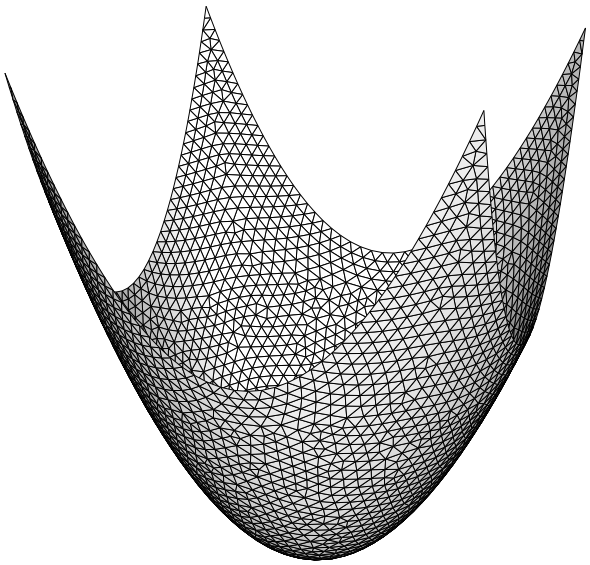


Figure 15: Paraboloid. Uniform default mesh.

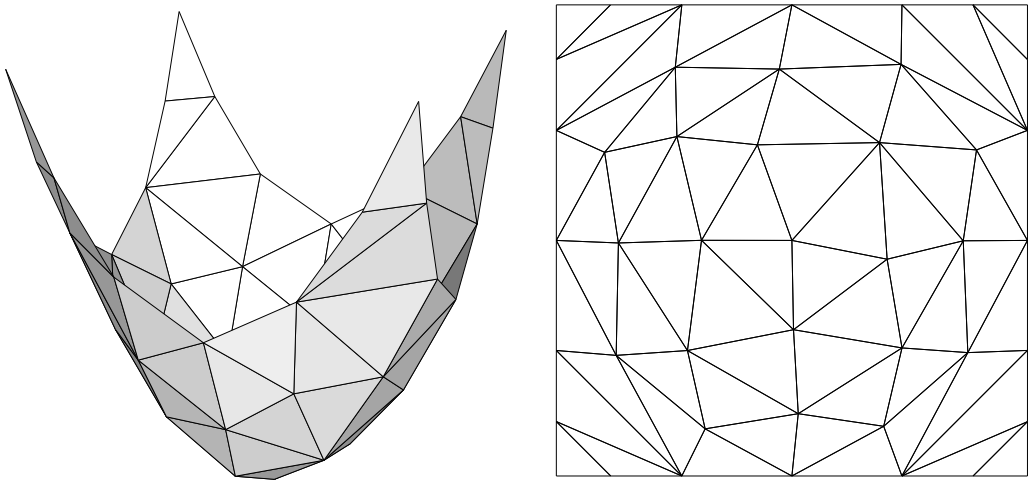


Figure 16: Paraboloid. Left, coarse uniform mesh with a central point. Right, mesh of the corresponding parametric domain.

File `x.pardom`:

```
1   ! ns (number of surfaces)

1   ! refs (reference of the surface)
1   ! orientation
4   ! nc (number of curves)
1 0 0 0   ! typc refc refp1 refp2
1 0 0 0
1 0 0 0
1 0 0 0
1   ! np (number of internal points)
1 0. 0.   ! refp u v
```

File `blsurf.env`:

```
hphydef 1.5
hphymax 1.5
call inimesh
call export

option onesurf2d
pref 2d
call export

option onesurf3d
pref 3d
call export

call exit
```

The next example consists in imposing internal lines.

In module `cad_m`, subroutine `cad_init` is modified in order to specify 20 straight segments and 16 circular arcs, and functions `curv0`, `curv1`, `curv2` implement the parameterization of the circular arcs:

$$\begin{array}{l} u = u_0 + r\ \cos\theta \\ v = v_0 + r\ \sin\theta \end{array} \quad \text{with} \quad \theta = \theta_1 + t\ (\theta_2 - \theta_1), \quad t \in [0,1]\,.$$

The following input files generate the 2D and 3D meshes shown in figure 17. With the option `flag` = 1, the discretization of the boundary and internal curves are better visualized. Each mesh contains 1772 vertices (+468 extremities caused by the curves discretization) and 3366 triangles.

File `x.pardom`:

```
1   ! ns (number of surfaces)

1   ! refs (reference of the surface)
1   ! orientation
36  ! nc (number of curves)
1 0 0 0   ! typc refc refp1 refp2
1 0 0 0
1 0 0 0
1 0 0 0
1 0 0 0
1 0 0 0
1 0 0 0
1 0 0 0
2 0 0 0
2 0 0 0
2 0 0 0
...
0   ! np (number of internal points)
```

File `blsurf.env`:

```
call inimesh

option allsurf2d
pref 2d
flag 1
call export

option allsurf3d
pref 3d
flag 1
call export

call exit
```

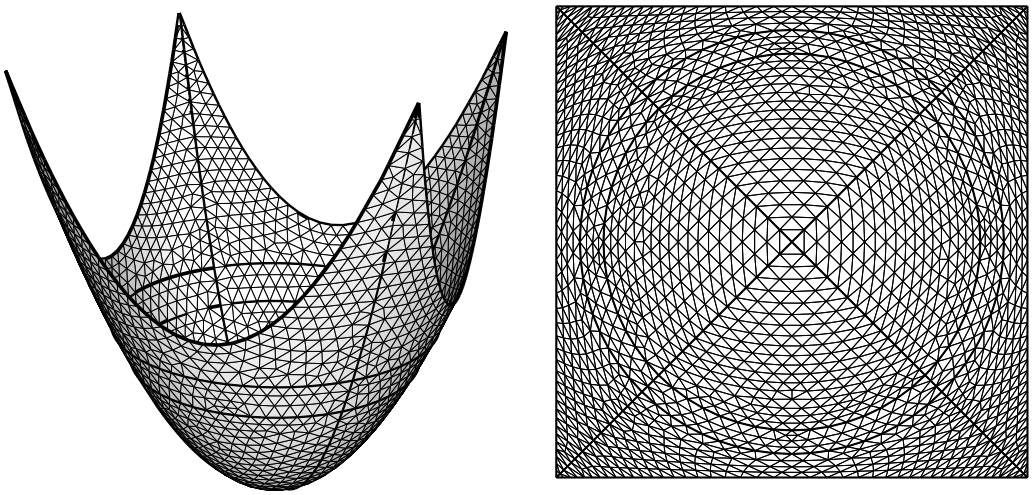


Figure 17: Paraboloid. Left, default uniform mesh with internal curves. Right, mesh of the corresponding parametric domain.

6.2 Utah teapot

We use a very familiar example, the teapot modeled by the University of Utah [10]. It is made up of 32 Bézier patches: 16 for the body (bottom included), 4 for the handle, 4 for the spout and 8 for the lid (see figure 18 left).

A bicubic Bézier patch is defined on $[0,1]^2$ by:

$$S(u,v) = \sum_{i=0}^{3} \sum_{j=0}^{3} P_{ij}\ B_i(u)\ B_j(v)\ ,$$

where $P_{ij},\ i = 0..3,\ j = 0..3$ is a given grid of control points, and $B_i(t),\ i = 0..3$ is the Bézier basis composed of the Bernstein polynomials:

$$\begin{aligned} B_0(t) &= (1-t)^3 \\ B_1(t) &= 3\ t\ (1-t)^2 \\ B_2(t) &= 3\ t^2\ (1-t) \\ B_3(t) &= t^3\ . \end{aligned}$$

We have modified the boundaries of the square $[0,1]^2$ so that the whole mesh of the body, the handle and the spout is conformal. To obtain this, we generated independently the surface meshes of different patches, computed the points at the intersections of these meshes, and finally obtained splines passing through these points with the de Boor algorithm [2].

To better illustrate the examples, we present meshes on the four patches that make up the front part of the body of the teapot. Figure 18 right shows the default uniform mesh. The length of the diagonal of the box bounding is $diag = 4.7$, hence the default physical size `hphydef` $= diag/50 = 0.094$. This mesh contains 1296 vertices and 2430 triangles.

With the following environment file, four meshes are exported, corresponding to the parametric domains with respective references 1, 4, 5 and 8. These meshes are shown in figure 19. At the top, vertical internal lines with abscissae $u = 0.3$ and $u = 0.8$ can be seen. They have been imposed to follow more closely the contour lines on the upper border.

File `blsurf.env`

```
call inimesh

option allsurf3d
pref 3d
call export

option onesurf2d
refs 1
pref 2d.1
call export

option onesurf2d
refs 4
pref 2d.4
call export

option onesurf2d
refs 5
pref 2d.5
call export

option onesurf2d
refs 8
pref 2d.8
call export

call exit
```

By adding both options `hphy_flag 0` and `hgeo_flag 1`, a geometric mesh is obtained (see figure 20). The indicated input files lead respectively to the default tolerance angle $\theta = 8°$ or a given angle $\theta = 16°$. Figure 20 shows these two meshes: left, 3293 vertices and 6482 triangles ; right, 1503 vertices and 2950 triangles. Some triangles are stretched because of the strong local variation of the radii of curvature, thus the sizes of the elements (which are proportional to the radii). In the next version of BLSURF, it will be possible to balance these size "shocks" [3].

The following example illustrates the specification of physical sizes (see figure 21). To this end, we give `hphy_flag` = 2 and the function `cad_hphys` returns respectively the sizes 0.05, 0.1, 0.2 and 0.8 on the four patches (of references 1, 4, 5 and 8). On the interface curves (extremities excepted), the prescribed sizes are the arithmetic means (0.8+0.1)/2 = 0.45, (0.1+0.05)/2 = 0.075, (0.05+0.2)/2 = 0.125 et (0.2+0.8)/2 = 0.5. At the common point of these four curves, the prescribed size is the arithmetic mean (0.45+0.075+0.125+0.5)/4 = 0.2875. Figure 21 shows, left, the resulting mesh (475 vertices and 843 triangles). By adding `hgeo_flag` = 1, the geometry of the object is considered also (right, 3460 vertices and 6784 triangles).

```
subroutine cad_hphys(refs, uv, h)
  integer :: refs
  double precision :: uv(2), h
  double precision, parameter ::    &
    th(8) = (/ 0.05, 0., 0., 0.1,   &
               0.2,  0., 0., 0.8 /)

  h = th(refs)
end subroutine cad_hphys
```

```
subroutine cad_hphyc(refs, ic, t, h)
  integer :: refs, ic
  double precision :: t, h
end subroutine cad_hphyc

subroutine cad_hphyp(refp, h)
  integer :: refp
  double precision :: h
end subroutine cad_hphyp
```

To illustrate the influence of the computations of the means, we return to the preceding example (figure 21 left). The environment variable `hmean_flag` has the default value 0, which corresponds to an arithmetic mean of the sizes. With `hmean_flag` = -1, the prescribed size on a curve is the minimum of those on the adjacent surfaces, thus producing the mesh in figure 22 left (534 vertices and 960 triangles). With `hmean_flag` = 1 (geometric mean), the mesh in figure 22 right is obtained (494 vertices and 881 triangles).

To illustrate now the influence of the interpolation of the prescribed sizes, we impose a size of 0.01 at the central point, 0.1 at the extremities of the upper curves, and 0.5 at the extremities of the lower curves. Figure 23 left corresponds to the linear interpolation obtained by default (633 vertices and 1188 triangles). Figure 23 right corresponds to a geometric interpolation (2748 vertices and 5414 triangles).

```
subroutine cad_hphys(refs, uv, h)
  integer :: refs
  double precision :: uv(2), h
end subroutine cad_hphys




subroutine cad_hphyc(refs, ic, t, h)
  integer :: refs, ic
  double precision :: t, h
end subroutine cad_hphyc
```

```
subroutine cad_hphyp(refp, h)
  integer :: refp
  double precision :: h

  if (refp == 6) then
     h = 0.01
  else if (refp <= 26) then
     h = 0.1
  else
     h = 0.5
  end if
end subroutine cad_hphyp
```

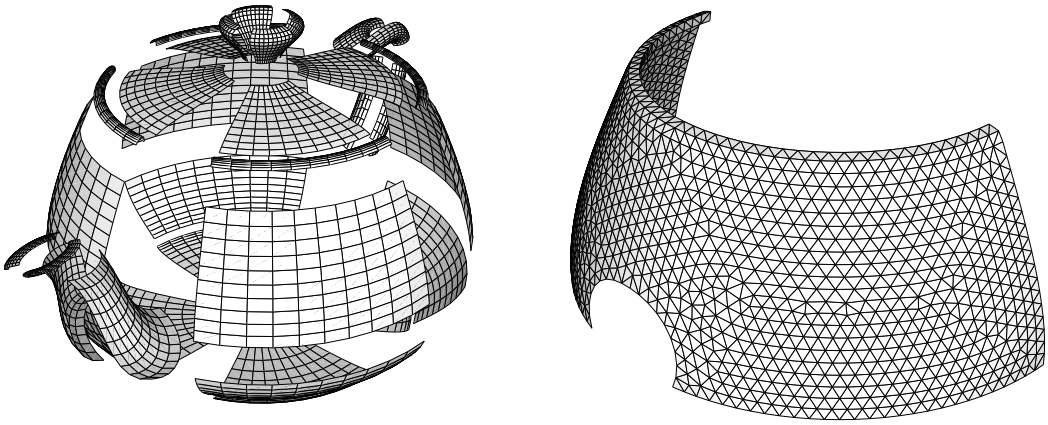


Figure 18: Utah teapot. Left, the 32 Bézier patches. Right, the default uniform mesh of four patches.

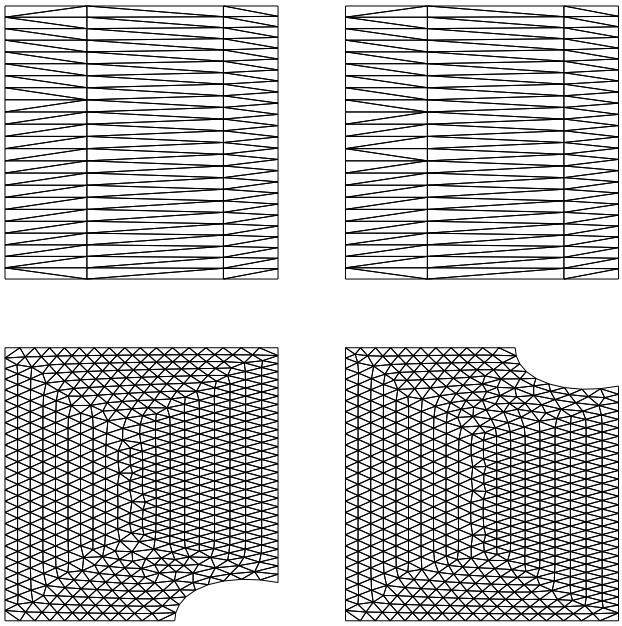
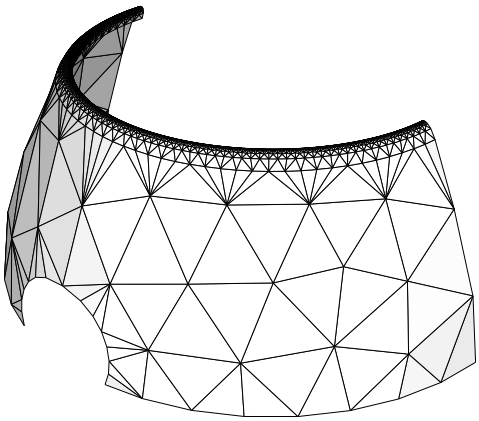
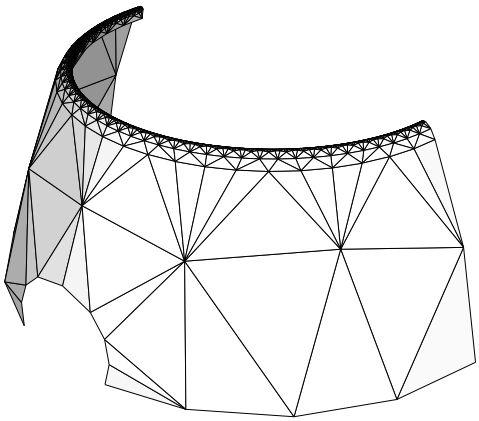


Figure 19: Utah teapot. Meshes of the four parametric domains corresponding to the previous uniform mesh.

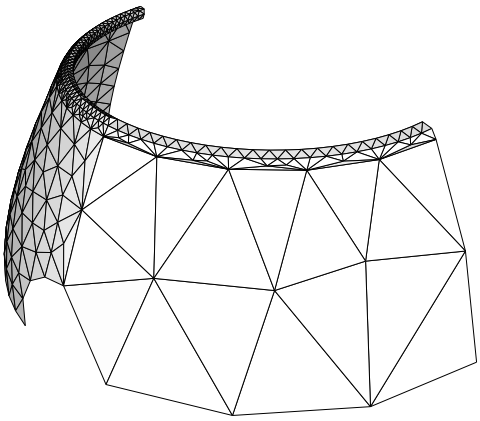


```
hphy_flag 0
hgeo_flag 1
```

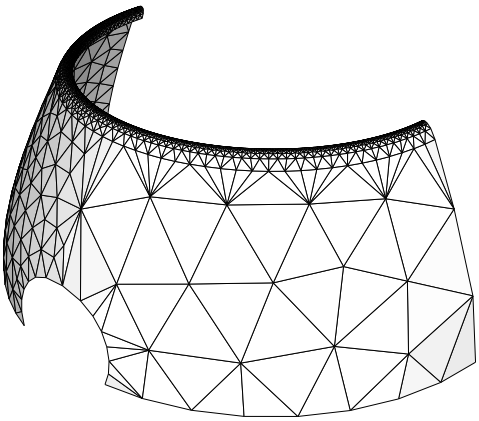


```
hphy_flag 0
hgeo_flag 1
angle_mesh 16
```

Figure 20: Utah teapot. Left, geometric mesh with the default angle $\theta = 8°$. Right, with an angle $\theta = 16°$.

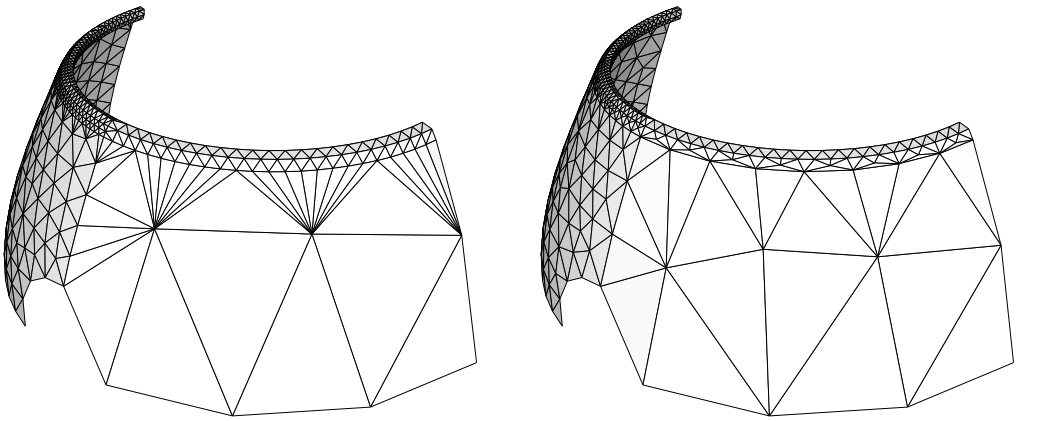


```
hphy_flag 2
```



```
hphy_flag 2
hgeo_flag 1
```

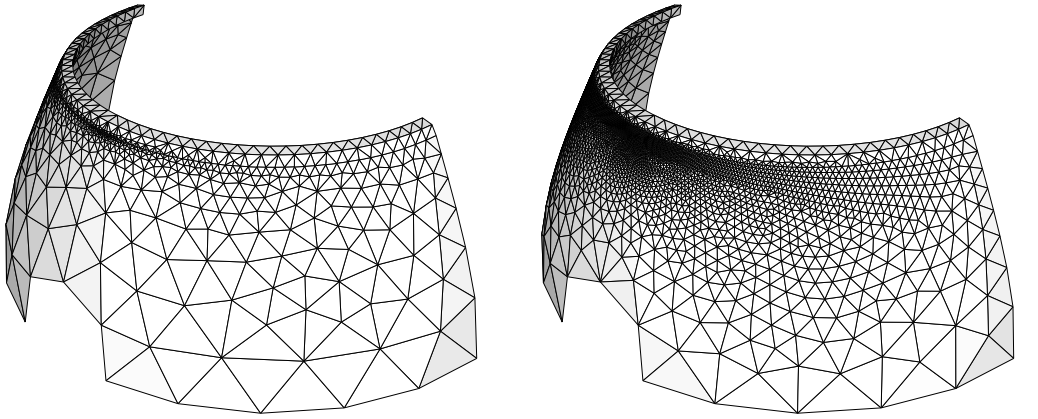
Figure 21: Utah teapot. Left, constant size for each patch. Right, mesh conforming to both the same prescribed sizes and the geometry.



```
hphy_flag 2
hmean_flag -1
```

```
hphy_flag 2
hmean_flag 1
```

Figure 22: Utah teapot. Prescribed sizes at the interfaces obtained with the minimum value (left) or with the geometric mean (right).



```
hphy_flag 2
hinterpol_flag 0
```

```
hphy_flag 2
hinterpol_flag 1
```

Figure 23: Utah teapot. Size interpolation: linear (left) or geometric (right).

To complete these examples, we show in figure 24 the mesh of the entire teapot (32 patches) with a constant elements size $h = 0.1$. This mesh contains 5917 vertices and 11658 triangles. For information, the CPU time was 29 seconds on a HP 9000/780 workstation, but this time can be greatly improved by optimizing the external functions (`surf0` and `surf1`) or by modifying the environment variables (for instance `LSS` or `frontal`).

File `blsurf.env`:

```
hphydef 0.1
call inimesh
call export
call exit
```

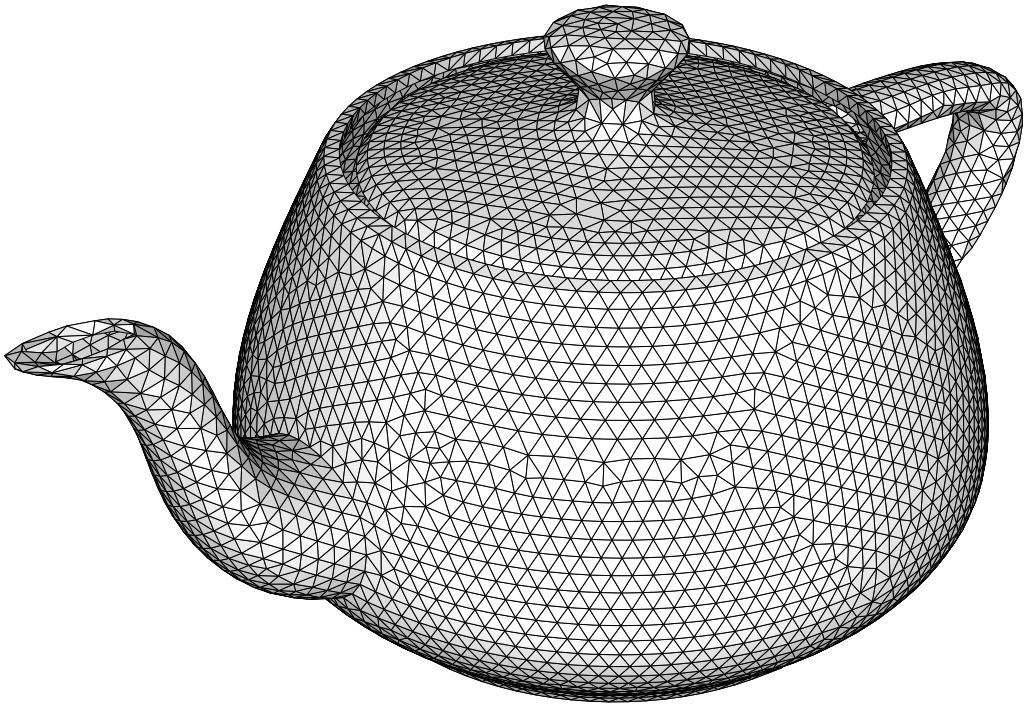


Figure 24: Utah teapot: uniform mesh.

6.3 Bust of Victor Hugo

By writing the appropriate external functions, the BLSURF software can be used to mesh discrete surfaces (defined by a structured grid). As an illustration, we have created an adapted mesh of a discrete cylindrical surface (the bust of Victor Hugo) obtained with a digitalization system "3D VideoLaser" [9] (cf. figure 25).

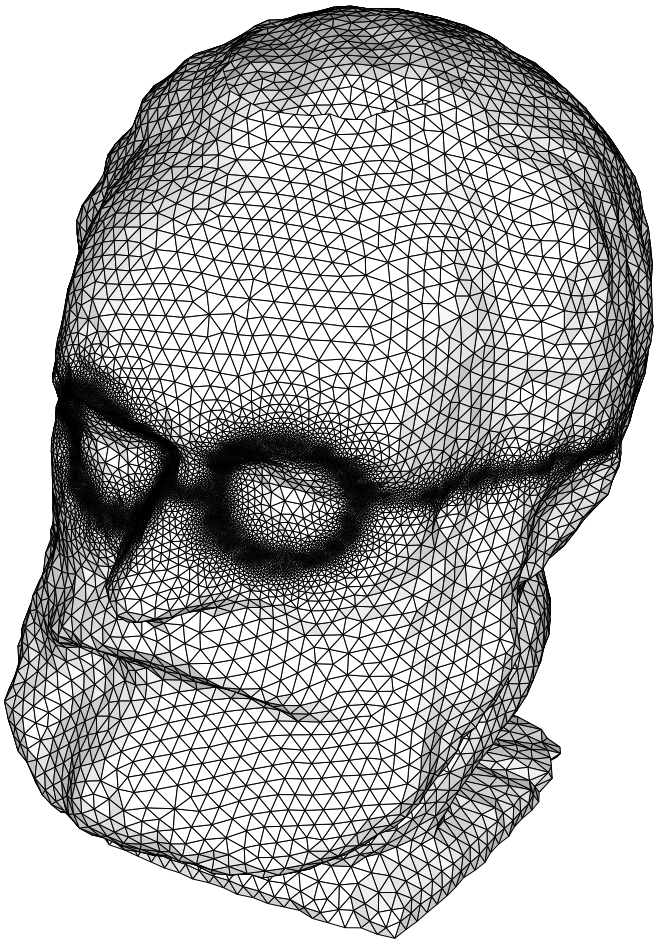


Figure 25: Bust of Victor Hugo (adapted mesh).

7 Conclusion and future extensions

The current BLSURF version has several restrictions, most of which have already been mentioned. Some of these will be removed within a short time:

- Although the memory is in general managed dynamically, the mesh of **each** patch is limited to 300 000 points and 600 000 triangles. If this limit is exceeded, the error message *"insufficient memory"* appears. It is then necessary to modify the source file of the bidimensional mesh generator. This will shortly be replaced by a more dynamic memory management.
- The generated elements are of type P1 only (three-node triangles). The next version will also generate elements of type P2 (six-node triangles), Q1 (four-node quadrilaterals) et Q2 (eight-node quadrilaterals).
- The next version will be able to balance the "h shocks" (strong variations of the prescribed sizes) [3].
- The next version will allow the user to prescribe sizes at the vertices of a background mesh.
- The next version will allow the user to specify an anisotropic map, i.e. the size and the shape of the elements.
- The auxiliary files (written by some components of BLSURF and read by others) will be replaced by associated arguments, which considerably reduces both CPU time and disk space.

Despite the above restrictions, it is already possible to integrate the BLSURF software into a CAD system, and to generate good quality surface meshes in a reasonable CPU time.

References

[1] M. Berger. *Géométrie – tome 2*. Nathan, 1990.

[2] C. de Boor, *A Practical Guide to Splines*, Springer, 1978.

[3] H. Borouchaki, F. Hecht, P. Frey. *Mesh Gradation Control*. International Journal for Numerical Methods in Engineering, vol. 43, n° 6, pp. 1143-1165, 30 November 1998.

[4] H. Borouchaki, P. Laug, P.L. George. *About parametric surface meshing*. 2^{nd} Symposium on Trends in Unstructured Mesh Generation, USNCCM'99, University of Colorado, Boulder, CO, USA, 4-6 août 1999.

[5] B.W. Char et al. *First Leaves: a Tutorial Introduction to MAPLE V*. Springer, 1992.

[6] Ch. Faure, Y. Papegay. *Odyssée User's Guide – Version 1.7*. Rapport Technique INRIA RT-0224, septembre 1998.

[7] P.L. George, H. Borouchaki. *Delaunay Triangulation and Meshing*. Ed. Hermès, Paris, 1998.

[8] P. Laug, H. Borouchaki. *Maillage de l'enveloppe d'une réunion de sphères*. Revue internationale de CFAO et d'informatique graphique, 13(1), pp. 43-64, mars 1998.

[9] F. Schmitt, H. Maître, A. Clainchard and J. Lopez-Krahm. *Acquisition and Representation of Real Object Surface Data*. SPIE Proceedings of Biostereometrics'85 Conference, vol. 602, Cannes, France, 2–6 December, 1985.

[10] A. Watt. *Fundamentals of Three-Dimensional Computer Graphics*. Addison-Wesley, 1989.

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399